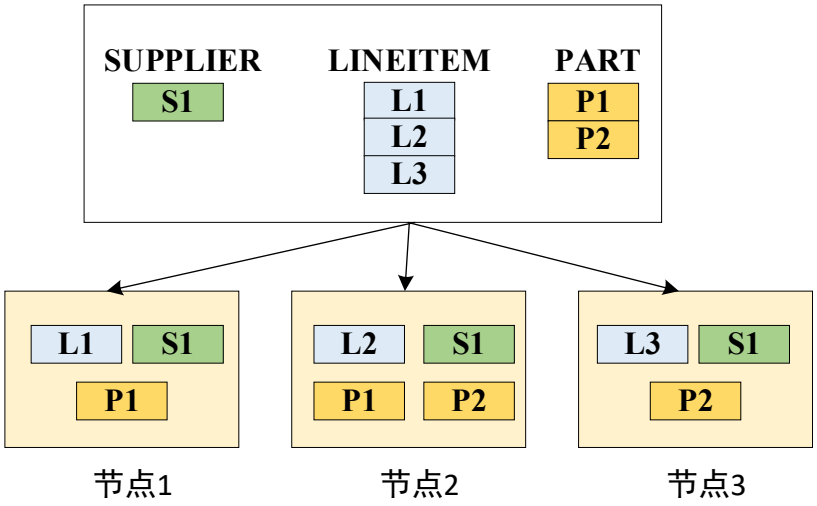
第八章

1.下图是分布式数据库中的三个关系 SUPPLIER、LINEITEM、PART 的数据分片与复制策略图解，请对该策略作出解读说明。



图中 SUPPLIER 为全复制，LINEITEM 为非复制，PART 为部分复制。设 LINEITEM 表包含 3 个分断 L1、L2 和 L3，SUPPLIER 表包含 1 个片段 S1，PART 表包含 2 个片段 P1 和 P2。LINEITEM 表的 3 个片段以非复制模式分别分配在节点 1、2、3 上，数据存储效率高，通过 3 个节点片段上的并行扫描提高 LINEITEM 表的并行数据访问性能，但当节点发生故障时则产生数据丢失问题；SUPPLIER 表的 1 个片段采用全复制方式分配在全部的 3 个节点上，与节点上 LINEITEM 表片段可以执行本地节点上的连接操作 $Li \bowtie Si$（i=1/2/3），提高并行连接性能，节点故障时可以由其它节点提供数据访问，但 SUPPLIER 表存储代价增加，而且更新需要在 3 个节点间同步，提高了数据更新代价；PART 表采用部分复制方式，存储效率和数据更新代价介于全复制与非复制之间，每个节点上的 PART 表片段与 LINEITEM 表片段没有键值对应关系，节点上 LINEITEM 表片段与 PART 表片段的连接操作需要访问其他节点的数据。PART 表片段的冗余存储策略支持当任何一个节点发生故障时，另外两个节点都能提供完整的 PART 表片段。

2.什么是分布式关系数据库的水平分片、导出分片、混合分片？如何构建关系的分片树？

水平分片根据元组来划分关系，每一个数据片段是关系元组的一个子集。设 R, R1, R2,…, Ri,…, Rn(1≤i≤n）为关系，若满足如下条件：

（1）$Ri \subseteq R$　（2）$R1 \cup R2 \cup Ri \cup \cdots \cup Rn=R$　（3）$Ri \cup Rj=\varnothing$（$i \neq j$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$）则称{R1, R2,…, Ri,…, Rn}为关系 R 的水平分片。

导出式水平分片是基于另一个关系的谓词对关系进行的水平分片。例如对关系 A 进行水平分片后，A 的主键 primary key 同时也是关系 B 的外键，于是关系 A 和关系 B 可以通过构建参照完整性关系进行连接。

混合分片通过垂直分片间的连接操作和水平分片间的集合并操作重构原始关系。

在构建关系的分片树时，通过根节点表示全局关系，叶子节点表示最后得到的片段关系，非叶节点表示分片过程的中间结果，边用 H 和 V 分别表示水平和垂直分片，节点名表示全局关系名和片段名。

3.如果关系 R 和 S 分别位于分布式数据库的两个节点，请写出通过半连接运算实现关系 R 和关系 S 的连接运算的关系代数表达式，并简要说明半连接运算在这样的连接运算中能减少网络通讯开销的理由。

关系 R 和 S 在属性 A 上的半连接可定义为：$R \ltimes S=R \bowtie \prod_A(S)$，$S \ltimes R=S \bowtie \prod_A(R)$
因此 R 和 S 通过属性 A 的连接操作可转化为：$R \bowtie S=(R \ltimes S) \bowtie S=(R \bowtie \prod_A(S)) \bowtie S$

半连接运算可分解为两部分操作：连接过滤、实际的记录连接。当连接过滤能较大缩减连接关系的记录数量时，先进行连接属性的连接过滤（半连接），再依据过滤结果传输实际关系子集能较好降低网络通讯代价。

第九章

1.（1）分布式数据库两阶段提交 2PC 算法的目的是什么？简述其基本过程。

2PC 算法的目的是保证提交阶段的原子性操作。其基本过程分为准备阶段和提交阶段两部分：

阶段 1：准备阶段（投票阶段）
（1）协调者节点在本地记录 Begin Commit 信息到 REDO 日志；
（2）协调者向所有参与者询问是否可以执行提交操作（发起投票），并等待所有参与者答复；
（3）各参与者检查是否可以提交，或者执行子事务操作直到提交前那一刻，将 undo 和 redo 信息记入子事务日志中（但不提交事务）。如参与者认为可以提交，将 Ready 写入本地 REDO 日志；
（4）参与者回应协调者，同意提交则返回 Vote Commit 消息，进入就绪状态等待协调者进一步消息；如果参与者本地失败，则返回 Vote Abort 消息，并写入 Abort 到本地日志。

阶段 2：提交阶段（完成阶段）
分为两种情况：
A.若协调者收到所有参与者的 Vote Commit：
（1）协调者在本地记录 Commit 信息到 REDO 日志；
（2）协调者向所有参与者节点发出 Global Commit 消息，进入 Commit 状态；
（3）参与者收到 Global Commit 消息后，记录 Commit 消息到 REDO 日志，正式完成提交操作（设置了事务提交完成标志），释放事务期间占用的资源；
（4）参与者向协调者发送 Commit End 消息；
（5）如果协调者收到了所有参与者的 Commit End 消息，在本地记录 End Commit 信息到 REDO 日志，完成事务。

B.若协调者收到某个参与者的 Vote Abort，或者协调者在第一阶段的询问超时之前无法获取某些参与者的回应：
（1）协调者在本地记录 Abort 信息到 REDO 日志；
（2）协调者向所有参与者节点发出 Global Abort 消息，进入 Abort 状态；
（3）参与者收到 Global Abort 消息后，记录 Abort 消息到 REDO 日志，利用事务回滚机制执行回滚操作，释放事务期间占用的资源；

（4）参与者向协调者发送 Abort End 消息；
（5）协调者收到了所有参与者的 Abort End 消息后，事务完成，在本地记录 End Abort 信息到 REDO 日志。

（2）2PC 提交算法的缺点有哪些？针对这些缺点目前有哪些改进的策略？

2PC 是一个阻塞性质的协议，有下述缺点：

①同步阻塞：参与者在等待其他参与者节点的响应过程中，所有的参与者节点都是事务阻塞的；

②单点故障：协调者一旦发生故障，参与者会一直阻塞下去，尤其在提交阶段，参与者都处于锁定事务资源的状态中；

③数据不一致：在提交阶段，当协调者向参与者发送 commit 请求后，发生了局部网络异常或在发送 commit 请求时协调者发生了故障，若只有部分参与者收到了 commit 请求，但其他部分未接到 commit 请求的节点无法执行提交操作，此时出现了数据不一致现象。

针对上述缺点，目前改进的策略包括三阶段提交（3PC）、基于 paxos 的 2PC 等改进协议。

2.（1）简述三阶段提交 3PC 算法的基本步骤；

基本步骤主要分为三部分：

阶段 1：canCommit

协调者向参与者发送 commit 请求，参与者如果可以提交就返回 Vote Commit 消息（参与者不执行事务操作），否则返回 Vote Abort 消息。

阶段 2：preCommit

协调者根据参与者的回答和超时机制，确定是否可以继续事务的 preCommit，分两种情况：

A.收到所有参与者的 Vote Commit，协调者发出 Global Commit 消息，进入下一阶段；

B.某个参与者返回 Vote Abort 或者等待超时，协调者发出 Global Abort 消息，参与者执行事务回滚。

阶段 3：doCommit

进行真正的事务提交。协调者向所有参与者发出 doCommit 请求，参与者收到该消息后正式执行事务提交，并释放事务期间占用的资源。各参与者向协调者反馈完成的 ack 消息，协调者收到所有参与者反馈的 ack 消息后，即完成事务提交。在这一阶段中，如果协调者出现问题或者协调者与参与者网络出现问题，都会导致参与者无法接收到协调者发出的 doCommit 请求。此时参与者会在等待超时（在消息传递过程中存在超时机制）之后，继续执行事务提交。

（2）相对于 2PC 提交，3PC 提交有哪些优点？

在等待超时后协调者或参与者会中断事务，相对于二阶段提交（2PC）减小了阻塞范围；避免了协调者单点问题，阶段 3 doCommit 中协调者出现问题时，参与者会继续提交事务。

第十章（仅作为学习思考题，不要求作业作答）

1.简述 paxos 算法的主要步骤，并说明其集群节点是如何交流数据的值的？

2.简述 raft 算法的主要步骤，并简要说明如何理解该算法是建立在 paxos 算法基础上的。